Connecting Global and Local Agent Navigation via Topology

Wouter van Toll, Julien Pettré

Connecting Global and Local Agent Navigation via Topology

Wouter van Toll
wouter.van-toll@inria.fr
Univ Rennes, Inria, CNRS, IRISA
Rennes, France

Julien Pettré
julien.pettre@inria.fr
Univ Rennes, Inria, CNRS, IRISA
Rennes, France

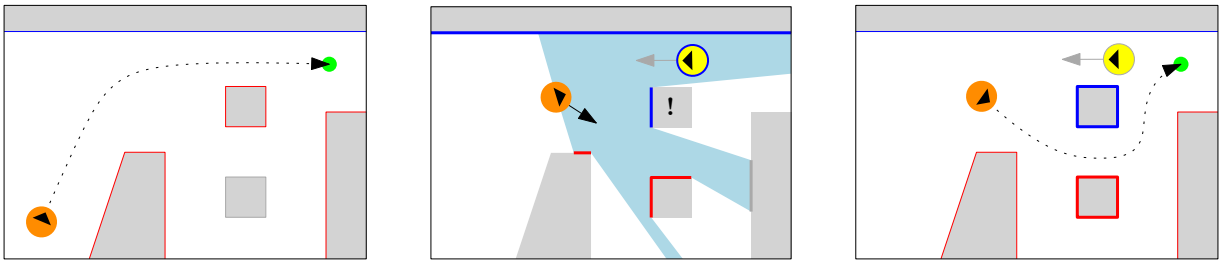


Figure 1: We define a *navigation strategy* as a set of decisions to pass obstacles and agents via the left (red) or right (blue). Left image: A global path yields a long-term strategy for obstacles. Middle image: A local velocity yields a short-term strategy for nearby agents and obstacles. For the obstacle marked with '!', the global and local strategies are in conflict. Right image: Our method detects these conflicts, and it allows the agent to plan a new global path that complies with its local strategy.

ABSTRACT

We present a novel topology-driven method for improving the navigation of agents in virtual environments. In agent-based crowd simulations, the combination of global path planning and local collision avoidance can cause conflicts and undesired motion. These conflicts are related to the decisions to pass obstacles or agents on certain sides. In this paper, we define an agent's navigation behavior as a topological strategy amidst obstacles and other agents. We show how to extract such a strategy from a global path and from a local velocity. Next, we propose a simulation framework that computes these strategies for path planning, path following, and collision avoidance. By detecting conflicts between strategies, we can decide reliably when and how an agent should re-plan an alternative path. As such, this work bridges a long-existing gap between global and local planning. Experiments show that our method can improve the behavior of agents while preserving real-time performance. It can be applied to many agent-based simulations, regardless of their specific navigation algorithms. The strategy concept is also suitable for explicitly sending agents in particular directions.

CCS CONCEPTS

• **Computing methodologies** → **Motion path planning**; **Intelligent agents**; **Real-time simulation**.

KEYWORDS

intelligent agents, path planning, navigation, crowd simulation

MIG '19, October 28–30, 2019, Newcastle upon Tyne, United Kingdom

ACM Reference Format:
Wouter van Toll and Julien Pettré. 2019. Connecting Global and Local Agent Navigation via Topology. In *Motion, Interaction and Games (MIG '19), October 28–30, 2019, Newcastle upon Tyne, United Kingdom.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3359566.3360084

1 INTRODUCTION

Simulating the motion of human crowds is a research topic with many real-world and entertainment applications [Pelechano et al. 2016; Thalmann and Musse 2013]. Many crowd-simulation techniques are *agent-based*: they model each member of the crowd as an intelligent agent with its own properties and goals. Within this paradigm, it is common to split an agent's navigation task into *global* path planning (finding an overall route to the goal) and *local* behavior (following this route while avoiding short-term collisions) [van Toll et al. 2015]. Recently, *mid-term* planning has been proposed as a step in-between [Bruneau and Pettré 2017]. As such, agents combine navigation algorithms for different levels of detail.

This multi-level navigation approach often works well, but it can have problems in scenarios with many agents or obstacles. The algorithms of different levels may give conflicting commands, for example when an agent's global path runs through a passage that turns out to be blocked by other agents (as in Figure 1(b)). The agent is then likely to get stuck, unless the global and local algorithms actively collaborate to plan a detour.

Although solutions (e.g. re-planning or waiting) may seem obvious to the human eye, it is difficult to detect automatically when global and local planning are in conflict. We argue that this is due to a design choice in local algorithms: in each simulation frame, their outcome is simply a velocity that optimizes local criteria, and not an *explicit decision* to pass agents or obstacles on a certain side.

1.1 Goals and Contributions

We present a novel topology-inspired approach to improve navigation in crowds. We model agents' global *and* local navigation plans as decisions to move around obstacles and other agents via the left or right. With this unified representation, we can systematically detect and resolve conflicts between the navigation decisions of different levels. This can yield more responsive agents and more efficient crowd motion. Figure 1 shows an example.

Planning a new global path (i.e. *re-planning*) is a relatively costly operation that one would prefer to avoid unless it is necessary. A conflict between global and local planning is an intuitive trigger for re-planning. By detecting these conflicts, we contribute to an efficient simulation where agents re-plan at the right time.

Our main contributions are the following:

- We formulate an agent's navigation behavior as a topological *strategy* amidst obstacles and other agents (Section 3.2).
- We show how to obtain such a strategy from a global path (Section 4.1) and from a local velocity (Section 4.2). This gives a *global* and a (dynamic) *local* strategy per agent.
- We suggest a simulation framework in which path planning, path following, and collision avoidance collaborate to detect and resolve conflicts between their strategies (Section 5).
- Via experiments (Section 6), we show that this conflict resolution can improve the crowd's behavior in several scenarios, without sacrificing real-time performance. We also show that our concepts can be used to easily guide agents in specific directions, e.g. to simulate the effect of signage in buildings.

We emphasize that we do *not* propose a new algorithm for mid-term planning or collision avoidance. Instead, we offer a fundamentally new way to harmonize agent-navigation algorithms, by assigning a uniform meaning to the choices made at each level. This concept can be applied to many different navigation algorithms. Thus, it can help improve a simulation regardless of its details.

2 RELATED WORK

In an agent-based crowd simulation, each member of the crowd plans a path to its own goal. While traversing their paths, agents use local algorithms to adjust their motion when needed. This *multi-level* simulation approach is the current state of the art, and many such systems have been developed in recent years [Curtis et al. 2016; Kielar et al. 2016; Pelechano et al. 2007; van Toll et al. 2015].

2.1 Global and Local Path Planning

2.1.1 Global Planning. To facilitate path planning in a 2D or 3D environment with obstacles, a *navigation mesh* subdivides the environment into regions [van Toll et al. 2016]. Many types of navigation meshes exist that can be computed automatically [Geraerts 2010; Kallmann 2014; Oliva and Pelechano 2013].

Given a navigation mesh, agents plan a path on the dual graph of the mesh regions, using graph search algorithms such as A* [Hart et al. 1968]. This results in a sequence of regions for the agent to move through. Within this sequence, agents can compute a specific geometric curve to follow. We will use the term *path* for this curve.

2.1.2 Local Planning. In each frame of the simulation loop, agents choose a point on their path to move to. This induces a *preferred velocity* $\mathbf{v}_{\text{pref}}$ per agent [Jaklin et al. 2013; Karamouzas et al. 2009]. Then, agents apply *collision avoidance* to find a velocity $\mathbf{v}_{\text{new}}$ that is (ideally) close to $\mathbf{v}_{\text{pref}}$ while avoiding nearby obstacles *and* agents. Collision avoidance is a popular research topic, and increasingly intelligent solutions have been proposed [Dutra et al. 2017; Helbing and Molnár 1995; van den Berg et al. 2011; Wolinski et al. 2016]. Local planning can also entail other tasks, such as modelling social groups [Kremyzas et al. 2016] and adapting an agent's motion to the local density [Best et al. 2014] or flow [van Goethem et al. 2015].

The term 'crowd simulation' is sometimes used for collision avoidance alone. However, path following and global planning are equally important, especially in environments with many obstacles. Small obstacles could be handled purely locally, but this only works up to a certain point that is hard to define. Therefore, we propose a solution where local and global planning can coordinate.

2.2 Other Navigation Techniques

2.2.1 Adding Levels or Information. Sometimes, a crowd is shaped in such a way that 'regular' global and local planning cannot bring an agent to its goal. There have been several attempts to improve the behavior in such cases. Examples are 'long-range collision avoidance' [Golas et al. 2013] where an agent prepares a rough plan to avoid agents that are far ahead, and 'mid-term planning' [Bruneau and Pettré 2017] where an agent refines its path to include the upcoming avoidance manoeuvres. The behavior of agents can also be improved by adding local information to global planning, such as the crowd density [Höcker et al. 2010; Pettré et al. 2007; van Toll et al. 2012] or the motion of dynamic objects [Kapadia et al. 2013]. While this adds intelligence, the different planning algorithms are still isolated processes that can produce conflicting strategies.

2.2.2 Hybrid Methods. Another option is to remove the global-local distinction by adding the agents as 'obstacles' to the navigation graph [Stüvel et al. 2016; Sud et al. 2007]. This is ideal for dense and (nearly) stationary crowds, where an agent should actively look for gaps between other agents. However, for other use cases, a combination of global and local planning generally works better.

Alternatively to agent-based methods, *flow-based* simulations model the crowd as a whole that moves along a flow field [Patil et al. 2010; Treuille et al. 2006]. These methods do not suffer from conflicts as much, but they cannot efficiently model crowds in which all agents have individual properties and goals. In this paper, we therefore focus on agent-based simulations.

2.2.3 Topology-Driven Navigation. All agent navigation methods, at every level of detail, have one common trait: they compute how an agent should move around objects (which can be agents or obstacles). In this paper, we propose the concept of a *navigation strategy* to formalize exactly this. It allows us to compare the navigation plans of different levels, and to resolve conflicts between them.

Our idea of enriching navigation with topology is not new. Robotics researchers have added topological constraints to global [Bhattacharya et al. 2012] and local planning [Knepper et al. 2011], and they have formulated multi-robot coordination in terms of topological decisions [Mavrogiannis and Knepper 2019]. Our work uses similar ideas, but focuses on the synchronization *between* levels of planning, and on the integration into real-time crowd simulations.

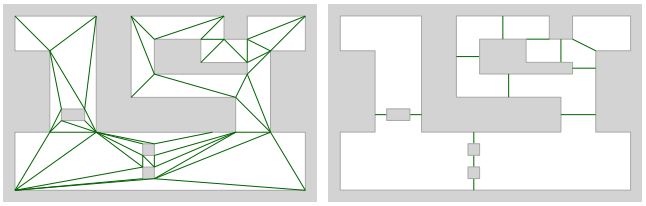


Figure 2: An environment with two possible navigation meshes. Passages are shown in green. Left: triangulation. Right: local minima of the medial axis, used in this paper.

3 DEFINITIONS

As in most crowd-simulation research, we approximate *agents* by disks. Let $AG = \{A_j\}_{j=0}^{m-1}$ be the set of m agents being simulated. All agents A_j are moving towards their individual goal positions $\mathbf{g}_j$, using any combination of navigation algorithms.

3.1 Environment and Navigation Mesh

We assume that the simulation takes place in a two-dimensional *environment* $\mathcal{E}$ with polygonal obstacles and a polygonal outer boundary, for which we will use the term 'obstacle' as well. Let $OBS = \{O_i\}_{i=0}^{n-1}$ be the set of n obstacles in $\mathcal{E}$, and let N be the total number of vertices for all obstacles. We assume that each obstacle O_i is a simple polygon, the inner obstacles do not overlap, and the outer boundary contains all inner obstacles. (If necessary, this obstacle representation can be obtained from a navigation mesh.)

To represent the environment for navigation purposes, we require a *navigation mesh* $\mathcal{M}$ that subdivides the obstacle-free space of $\mathcal{E}$ into non-overlapping polygonal regions $REG = \{R_i\}_{i=0}^{r-1}$. For any region R_i, all vertices should be on the boundary of an obstacle. Consequently, each boundary segment of R_i either lies on an obstacle boundary or is shared with another region R_j. We will use the term *passage* for a boundary segment shared by two regions. Each passage is a line segment that starts and ends at an obstacle.

Figure 2 shows two possible navigation meshes for an example environment. Many types of navigation meshes exist, including ones where the combined complexity of all regions is $O(N)$.

3.2 Decisions and Strategies

We now formalize the concept of a *navigation strategy*: a set of decisions to pass obstacles and agents on a certain side. Later, we will use this definition to assign a common topological meaning to the results of global and local planning.

3.2.1 Decision. During the simulation, any agent A_j that has not yet reached its goal should navigate around obstacles and other agents, to which we will collectively refer as *objects*. A_j can decide to pass any object via the left or via the right. Formally, let $D(A_j, B) \in \{\mathsf{L}, \mathsf{R}, \mathsf{X}\}$ be the *decision* of A_j with respect to an object B:

- $D(A_j, B) = \mathsf{L}$ if A_j intends to pass B via the *left* (thus keeping B on their right-hand side);
- $D(A_j, B) = \mathsf{R}$ if A_j intends to pass B via the *right* (thus keeping B on their left-hand side);
- $D(A_j, B) = \mathsf{X}$ if A_j has not (yet) made a decision for B.

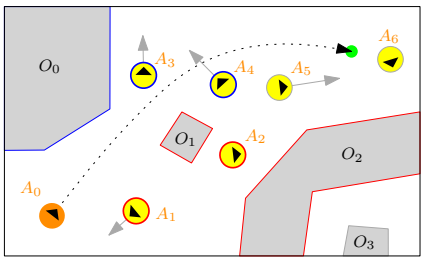


Figure 3: A navigation strategy. The agent A_0 will move to its goal (green) by passing obstacles and other agents, either via the right (outlined in blue) or via the left (outlined in red). Undecided or irrelevant objects are outlined in gray.

In the figures of this paper, we will use the color red to denote the decision L, blue to denote the decision R, and gray to denote X. Decisions of X do not have to be stored explicitly.

3.2.2 Strategy. A *navigation strategy* for A_j is a set of decisions with respect to all objects, according to a decision function D_X:

$$S(D_X, A_j) = \{D_X(A_j, B)\}_{B \in OBS \cup AG - \{A_j\}}$$

In terms of coloring, a strategy is an assignment of the colors red and blue to the objects between which the agent wants to pass, while leaving any irrelevant (or undecided) objects in gray. Figure 3 shows an example of a strategy. Here, note that the decision regarding a neighboring agent A_k also depends on the *velocity* of A_k. For instance, the agent A_4 is blue because it will have moved to the left before A_0 passes it. We will explain this further in Section 4.2.

In this paper, we will abbreviate $S(D_X, A_j)$ to S_X (for any subscript X). Thus, a strategy S_X always has a corresponding decision function D_X, and it will implicitly always concern A_j.

In a typical crowd simulation, each agent A_j tries to follow a global path. This path has a corresponding *global* strategy S_G. It usually does not yet describe how to avoid agents, i.e. $D_G(A_j, A_k) = \mathsf{X}$ for all other agents A_k. Over time, local navigation algorithms should determine how to move around agents. This results in one or more *local* strategies (one for each algorithm) that change over time. These local strategies and S_G might be in conflict, as they also impose their own decisions for nearby *obstacles*.

3.2.3 Comparing Strategies. To solve navigation problems, it is useful to define consistency between strategies. First, we say that two *decisions* $D_1(A_j, B)$ and $D_2(A_j, B)$ for a given object B are *inconsistent* if one decision is L and the other is R. Otherwise (i.e. if they are equal or if either decision is X), the decisions are *consistent*. We will use $=_c$ to denote consistency and $\neq_c$ for inconsistency.

Now, let S_1 and S_2 be two full *strategies*. For any object B where $D_1(A_j, B) \neq_c D_2(A_j, B)$, we say that S_1 and S_2 have a *conflict*. The two strategies are *consistent* if and only if they have no conflicts, i.e. $S_1 =_c S_2$ iff $D_1(A_j, B) =_c D_2(A_j, B)$ for all objects B. Otherwise, the strategies are *inconsistent*. We also introduce the notation $S_1 =_{OBS} S_2$ and $S_1 =_{AG} S_2$ for when two strategies have no conflicts regarding obstacles and agents, respectively.

The problem from Figure 1 occurs when the strategies of collision avoidance and path following (say, S_C and S_F) have an obstacle-related conflict. Our goal is to detect and resolve these situations.

4 OBTAINING NAVIGATION STRATEGIES

This section explains how to obtain navigation strategies from a *path* and from a *velocity*. Section 5 will use this to handle conflicts between path planning, path following, and collision avoidance.

4.1 Converting a Path to a Strategy

A global planning algorithm computes a *path* from a start position $\mathbf{s}$ to a goal position $\mathbf{g}$. This path is a curve $\pi : [0, 1] \rightarrow \mathbb{R}^2$ through the environment, where $\pi(0) = \mathbf{s}$ and $\pi(1) = \mathbf{g}$. The curve does not intersect any obstacles in *OBS*, but it may intersect the agents in *AG* because agents are (usually) not yet considered in this phase.

We now describe how to obtain a navigation strategy S from a path π. In other words, we show how π leads to a red/blue coloring of obstacles. Figure 4 shows an example of a path and its strategy. Note that the path keeps blue obstacles to its left and red obstacles to its right, while irrelevant obstacles remain gray.

4.1.1 Overview. To convert π to a strategy S, the first step is to find the sequence of *passages* that π traverses in the navigation mesh $\mathcal{M}$. In this paper, we assume that $\mathcal{M}$ was already used to compute π in the first place. The sequence of passages for π can then easily be constructed during the path-planning algorithm itself. (If π is an arbitrary curve that was not computed using $\mathcal{M}$, one could compute the passages that π intersects, and remove any duplicate entries caused by backtracking or loops in the path.)

Let $\mathcal{P} = \{p_i\}_{i=0}^{k-1}$ be the sequence of k passages that π traverses. The example path in Figure 4 visits nine passages. Recall that each passage p_i is a line segment with an obstacle at its left and right endpoint; let us denote these obstacles by $O_{i,l}$ and $O_{i,r}$ respectively. By traversing p_i, the agent will pass $O_{i,l}$ via the right (blue) and $O_{i,r}$ via the left (red). To compute all relevant navigation decisions (i.e. to define a decision function D), we check the traversed passages one by one, and we perform the following steps for each p_i:

(1) Try to set $D(A_j, O_{i,l})$ to R; that is, try to give $O_{i,l}$ the color blue. (We will explain below what 'trying to set' means.)
(2) Try to set $D(A_j, O_{i,r})$ to L; that is, try to make $O_{i,r}$ red.
(3) If p_i is not the last passage of $\mathcal{P}$, let R_i be the region to which p_i leads. In the clockwise ordering of passages bounding R_i, visit all passages between p_i and p_{i+1}. The light-blue passage in Figure 4 is an example of this case. For each such passage p', get its left obstacle O'_l and try to set $D(A_j, O'_l)$ to R.
(4) Symmetrically, visit all passages in the counter-clockwise ordering between p_i and p_{i+1}. Examples are the pink passages in Figure 4. For each such passage p', get its right obstacle O'_r and try to set $D(A_j, O'_r)$ to L.

This results in a strategy S, implicitly augmented with $D(A_j, B) =$ X for any undecided object B. If the boundaries of regions can be traversed via pointers (next/prev/twin), we can compute S in $O(N')$ time, where N' is the total complexity of the regions that π visits.

4.1.2 Handling Ambiguities. We use the term 'try to set' instead of 'set' because an obstacle O might occur as both a left and right obstacle at different points along the path. The outer boundary of Figure 4 is an example of this. In such cases, we want to set $D(A_j, O)$ to X, as we cannot use either L or R along the whole path. So, in the discussion above, 'try to set $D(A_j, O)$ to R' is defined as follows:

- If $D(A_j, O)$ has not yet been set before, set it to R.
- If $D(A_j, O)$ was already set to L, set it to X.
- If $D(A_j, O)$ was already set to R or X, then it stays that way.

The definition for the decision L is symmetric.

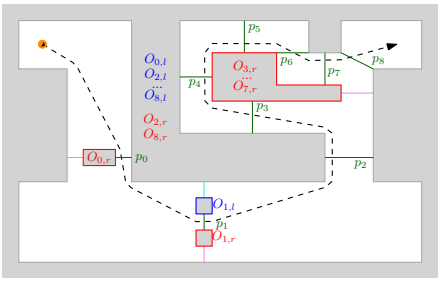


Figure 4: A global path (the dashed curve) and its navigation strategy, obtained from the relevant passages (in green) and their neighboring passages (in light blue and pink).

4.2 Converting a Velocity to a Strategy

In each simulation frame, every agent A_j performs path following and collision avoidance to compute (respectively) a preferred velocity $\mathbf{v}_{\text{pref}}$ and a new velocity $\mathbf{v}_{\text{new}}$. To detect conflicts, we need to convert these velocities to strategies as well.

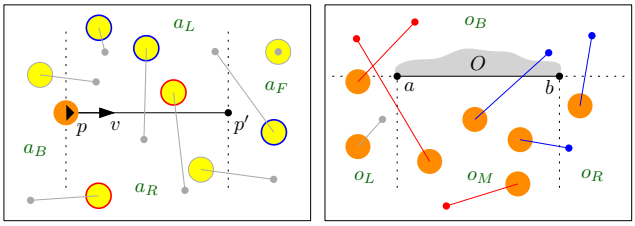
We now explain how to obtain a navigation strategy S for an arbitrary velocity $\mathbf{v}$. That is, we show how to compute a red/blue coloring of nearby objects assuming that agent A_j uses the velocity $\mathbf{v}$ for a certain amount of time. We focus on non-zero velocities. If $||\mathbf{v}|| = 0$, the agent A_j does not really 'navigate', and we can (implicitly) use $D(A_j, B) =$ X for all objects B.

In the following discussion, $\overline{\mathbf{ab}}$ denotes the line segment between two points $\mathbf{a}$ and $\mathbf{b}$. Let $\mathbf{p}$ be the current position of agent A_j, and let τ be the *time window*, a parameter that we will set in Section 5. In τ seconds, the velocity $\mathbf{v}$ will send A_j to position $\mathbf{p}' = \mathbf{p} + \mathbf{v} \cdot \tau$.

4.2.1 Neighboring Objects. The short-term use of a velocity only affects objects that are nearby. Similarly to collision-avoidance algorithms, we consider nearby *agents* and nearby *obstacle segments*. Thus, we treat the boundary segments of obstacles as separate entities. Appendix A specifies how we find these neighboring objects in our implementation. For each neighboring object, we determine a navigation decision. The final strategy S consists of all these decisions, (again) implicitly augmented with X for unhandled objects.

4.2.2 Decision for an Agent. Figure 5(a) shows how the segment $\overline{\mathbf{pp}'}$ divides the vicinity of A_j into four areas: back (a_B), front (a_F), left (a_L), and right (a_R). For a neighboring agent A_k with position $\mathbf{p}_k$ and velocity $\mathbf{v}_k$, let $\mathbf{p}'_k = \mathbf{p}_k + \mathbf{v}_k \cdot \tau$ be the predicted position of A_k after τ seconds. We define the decision $D(A_j, A_k)$ as follows:

- If $\mathbf{p}_k$ lies in a_B, then A_k starts behind A_j, so A_j does not actively pass it (anymore). Thus, $D(A_j, A_k) =$ X.
- If $\mathbf{p}'_k$ lies in a_F, then A_k ends up in front of A_j, so A_j does not pass it (yet). Thus, $D(A_j, A_k) =$ X.
- Otherwise, if $\overline{\mathbf{pp}'}$ and $\overline{\mathbf{p}_k\mathbf{p}'_k}$ intersect at some point $\mathbf{x}$, let t be the time after which A_j will reach $\mathbf{x}$. The decision depends on the position of A_k at that time, i.e. on $\mathbf{p}''_k = \mathbf{p}_k + \mathbf{v}_k \cdot t$:
  - If $\mathbf{p}''_k$ lies exactly on $\overline{\mathbf{pp}'}$, then $D(A_j, A_k) =$ X.



(a) Agent-Agent (b) Agent-Obstacle

Figure 5: A velocity v and its navigation strategy, with respect to (a) other agents, and (b) an obstacle segment.

- If $\mathbf{p}_k''$ lies in a_L, then $D(A_j, A_k) = \mathsf{R}$.
- If $\mathbf{p}_k''$ lies in a_R, then $D(A_j, A_k) = \mathsf{L}$.

- Otherwise, $\overline{\mathbf{p}_k\mathbf{p}_k'}$ lies fully or partly in either a_L or a_R.
  - If $\overline{\mathbf{p}_k\mathbf{p}_k'}$ overlaps with a_L, then $D(A_j, A_k) = \mathsf{R}$.
  - If $\overline{\mathbf{p}_k\mathbf{p}_k'}$ overlaps with a_R, then $D(A_j, A_k) = \mathsf{L}$.

Our definition ignores whether A_j and A_k are on collision course. Even if $\mathbf{v}$ would lead to a collision, it is useful to know which navigation decision A_j is most likely to have in mind.

4.2.3 Decision for an Obstacle Segment. For obstacle segments, different segments of the same obstacle O may give conflicting decisions. When this happens, we would like to set $D(A_j, O)$ to X, as we cannot decide uniformly for the entire obstacle. Therefore, just like in Section 4.1, we treat each obstacle segment either by ignoring it or by *trying* to set the decision for the corresponding object O, where 'trying to set' is defined as before.

For a segment $\overline{\mathbf{ab}}$ belonging to an obstacle O, assume (w.l.o.g.) that $\mathbf{a}$ and $\mathbf{b}$ appear on O's boundary in counterclockwise order. $\overline{\mathbf{ab}}$ divides the space into four areas, as shown in Figure 5(b): back (o_B), left (o_L), middle (o_M), and right (o_R). We treat $\overline{\mathbf{ab}}$ as follows:

- If $\mathbf{p}$ lies in o_B, ignore this segment: it is currently not visible to the agent A_j.
- If $\overline{\mathbf{pp'}}$ stays entirely inside o_L or o_R, ignore this segment: A_j does not make an active decision yet.
- If $\overline{\mathbf{pp'}}$ and $\overline{\mathbf{ab}}$ intersect, A_j will collide with $\overline{\mathbf{ab}}$. The decision depends on the angle θ between $\mathbf{v}$ and $\mathbf{b} - \mathbf{a}$:
  - If $\theta < 90°$, try to set $D(A_j, O)$ to R.
  - If $\theta > 90°$, try to set $D(A_j, O)$ to L.
  - If $\theta = 90°$, ignore this segment.
- Otherwise, if $\overline{\mathbf{pp'}}$ moves into o_B, let $\mathbf{x}$ be the first point where this happens.
  - If $\mathbf{x}$ is closer to $\mathbf{a}$ than to $\mathbf{b}$, try to set $D(A_j, O)$ to L.
  - Otherwise, try to set $D(A_j, O)$ to R.
- Otherwise, A_j moves along the segment without fully passing it. The decision then depends on θ as described earlier.

5 STRATEGY-DRIVEN SIMULATION

We now extend a generic crowd-simulation framework [van Toll et al. 2015] with navigation strategies to detect and resolve conflicts. In the existing framework, each agent A_j has three navigation tasks:

- Path planning (performed once): Compute a global path π to the agent's goal position $\mathbf{g}_j$.
- Path following (performed in each frame): First compute a *reference point* $\mathbf{p}_{\text{ref}}$ that denotes the agent's progress along π. Then compute an *attraction point* $\mathbf{p}_{\text{att}}$ on π, and a preferred velocity $\mathbf{v}_{\text{pref}}$ that sends A_j to $\mathbf{p}_{\text{att}}$ at its preferred speed s_{pref}.
- Collision avoidance (performed in each frame): Compute a velocity $\mathbf{v}_{\text{new}}$ close to $\mathbf{v}_{\text{pref}}$ that avoids nearby collisions.

This framework is not restricted to specific algorithms for path planning, path following, or collision avoidance. Our implementation uses the data structures and algorithms listed in Appendix A.2.

It is common to let an agent re-plan its path whenever the agent cannot compute an attraction point $\mathbf{p}_{\text{att}}$, i.e. when it no longer knows how to follow its current path [Jaklin et al. 2013]. We will do this in our implementation as well.

5.1 Overview of the Strategic Level

In the standard framework with three tasks, agents always accept the choices made by each algorithm. We propose to add a fourth task –the *strategic level*– in which the agent analyzes the strategies of each task and responds to conflicts.

In this paper, we focus on *obstacle*-related conflicts between path following and collision avoidance. In future work, we intend to add a form of mid-term planning, which will be another navigation level with its own strategy S_M. It will then make sense to also consider *agent*-related conflicts between S_M and the other strategies.

The strategic level first converts the agent's preferred velocity $\mathbf{v}_{\text{pref}}$ to a strategy S_F, and $\mathbf{v}_{\text{new}}$ to a strategy S_C, using the method from Section 4.2. We use a time window τ of $\frac{||\mathbf{p}_{\text{att}} - \mathbf{p}||}{s_{\text{pref}}}$; this is the time required to reach the attraction point at the preferred speed.

Next, we check if $S_C \neq_{OBS} S_F$. When this happens, collision avoidance and path following disagree on how to pass one or more obstacles. The agent is about to move around these obstacles differently than prescribed by $\mathbf{v}_{\text{pref}}$. We compute two ways to resolve the conflict: an *alternative path* π' that satisfies S_C (instead of S_F), and an *alternative velocity* $\mathbf{v}'_{\text{new}}$ that satisfies S_F (instead of S_C). Section 5.2 will describe how to compute π' and $\mathbf{v}'_{\text{new}}$.

When π' and $\mathbf{v}'_{\text{new}}$ have been computed, the agent can choose one of the two, or it can decide to ignore the conflict and use the velocity $\mathbf{v}_{\text{new}}$ that it already had in mind. There are many ways to let an agent choose between these three options. We employ a simple choice model with two parameters: a maximum detour factor W, and a Boolean flag *Strict*. The agent decides as follows:

(1) Use π' if it exists and if it is $\leq W$ times longer than the remainder of π. We define this remainder as the line segment from $\mathbf{p}$ to $\mathbf{p}_{\text{att}}$, plus the subpath of π from $\mathbf{p}_{\text{att}}$ to the goal $\mathbf{g}_j$.
(2) Otherwise, if *Strict* = `True`, use $\mathbf{v}'_{\text{new}}$.
(3) Otherwise, use $\mathbf{v}_{\text{new}}$.

5.2 Finding an Alternative Path and Velocity

We now explain how to compute a path or a velocity under the topological constraints of another strategy. Let the *constraint strategy* S_T be a strategy that contains all constraints of our interest.

5.2.1 Path Planning with Constraints. To plan a global path whose strategy $S_{G'}$ is consistent with S_T, we plan a path on the navigation mesh $\mathcal{M}$ in the usual way, but with the additional rule that we cannot traverse any passages that would lead to a conflict between

$S_{G'}$ and S_T. For any passage p_i that we encounter during the search, traversing p_i would imply $D_{G'}(A_j, O_{i,l}) = \texttt{R}$ and $D_{G'}(A_j, O_{i,r}) = \texttt{L}$. Thus, if $D_T(A_j, O_{i,l}) = \texttt{L}$ or $D_T(A_j, O_{i,r}) = \texttt{R}$, we do not search further in this direction, as traversing p_i would imply $S_{G'} \neq_c S_T$. (Technically, $D_{G'}(A_j, O_{i,l})$ could be cancelled out to X elsewhere on the path, but in this case, we assume that it is undesirable to pass an obstacle on the wrong side at least once.)

5.2.2 Collision Avoidance with Constraints. To perform collision avoidance such that the resulting strategy $S_{C'}$ is consistent with S_T, we should disallow the selection of any velocity that would lead to conflicts. If the collision-avoidance algorithm uses sampling (i.e. choosing the best velocity out of several candidates), then S_T is easy to incorporate. For each candidate velocity $\mathbf{v}''$, we compute the navigation strategy $S_{C''}$ and we discard $\mathbf{v}''$ if $S_{C''} \neq_c S_T$. We expect that a similar adaptation can be made for other types of collision-avoidance algorithms, such as those based on velocity obstacles [van den Berg et al. 2011].

5.2.3 Application. In the strategic level of our simulation framework (Section 5.1), we apply these adapted algorithms as follows:

- To compute the alternative velocity $\mathbf{v}'_{\text{new}}$, we perform collision avoidance constrained by S_F.
- To compute the alternative path π', we perform path planning constrained by S_C. To prevent an agent from backtracking, we augment S_C with the navigation decisions from the last 3 passages that the agent has traversed.

As mentioned in Section 1, global (re-)planning is a relatively costly operation. By planning an alternative path only in case of a conflict, we avoid re-planning when the current path is still attractive. Still, conflicts may occur often, and it is useful to let agents re-plan at most once every few seconds. Therefore, we introduce the *re-planning time* T_R: an agent only looks for an alternative path π' when its last path update was at least T_R seconds ago. In any frame where this is not the case, we consider π' to be non-existent.

6 EXPERIMENTS AND RESULTS

We have implemented our strategy-driven simulation framework in C++. All relevant settings and implementation details can be found in Appendix A. We perform all experiments on a Windows 10 device with an Intel Core i7-7920HQ CPU. Throughout this section, we compare the following variants of the simulation framework:

- *CA-Only*: agents perform no path planning or path following, and their $\mathbf{v}_{\text{pref}}$ always points straight to the goal.
- *Standard*: agents perform the three standard navigation tasks, but they do not compare any topological strategies.
- *Strategic*: the full strategy-driven framework from Section 5. We will test different values of the parameters T_R and W. (Note that *Strategic* can reproduce the behavior of *Standard* by using $T_R = \infty$, $W = 0$, and *Strict* = False.)

Our main purpose is to show how the strategic level can improve the behavior of agents compared to a standard simulation. We will show this via several scenarios, and we will discuss the results per scenario. Section 7 will discuss the advantages and limitations of our method in general. To see all scenarios in motion, we encourage the reader to watch this paper's *supplementary video.*

6.1 Single-Agent Demonstration

6.1.1 Experiment. To demonstrate the advantages of the strategic level, we show an agent navigating through an environment with obstacles and other agents. Figure 6(a) shows our example scenario. The red agent needs to move from the top left to the top right, and its initial path (computed by our path-planning method) runs through several problematic areas. Some passages are blocked by stationary agents (in purple and green), and the lower part contains agents that are flowing to the left (in blue) and to the right (in orange). These 'other' agents all use the *CA-Only* simulation variant.

6.1.2 Discussion of Results. Figure 6(b) shows the trajectories for the red agent when using different simulation variants. When the agent uses *CA-Only* (shown in pink), it gets trapped behind the first wall on the straight path to the goal. When it uses *Standard* (shown in brown), it gets stuck at the opening blocked by purple agents.

The red trajectory is the result for *Strategic* with $T_R = 2$, $W = \infty$, and *Strict* = False. That is, the agent re-plans at most every second, it accepts detours of any length, and it is not forced to follow its path locally. With these settings, the agent reaches the goal. Figure 6(c) shows where the agent changes its global plan. In cases 1 and 4, the agent chooses a detour because collision avoidance recognizes that an area is blocked. In cases 2 and 3, the agent switches between different openings and eventually chooses the lowest one, to join the orange agents who are moving in the same direction.

Note that this is merely meant as a demonstration of how the strategy-based simulation *could* be used. Many variations are possible. For example, if we used $W = 1$ instead, the agent would consider the last detour to be too large, and it would (deliberately) keep trying to use the passage with the green agents. This may be the desired behavior in certain scenarios.

6.2 Crowd Flows amidst Small Obstacles

6.2.1 Experiment. To show what happens when many agents use a certain simulation variant, we simulate unidirectional crowd flows in a U-turn corridor with small obstacles (*UTurn*, 40 × 24 m). We compute start and goal positions via uniform random sampling at the corridor's far ends. We insert 3 agents per second, and we remove an agent when it lies within 0.5m of its goal. To make the crowd more diverse, we give each agent a (uniformly sampled) random preferred speed s_{pref} between 1.2 and 1.4m/s. Figure 7 shows still images of this scenario with different simulation variants.

We measure the total number of agents (*NrAgents*) that are added after $t = 50$s and that reach their goal before $t = 300$s. For each such agent, we measure the distance travelled, the total travel time, the average speed, and the time spent walking more slowly than 0.5m/s. Of these metrics, we report the average and standard deviation over all agents (*Dist*, *Time*, *AvgSpeed*, *SlowTime*). These are indicators of how efficiently the crowd moves. We exclude the first 50s to prevent distorting the results with a quiet 'warm-up' period.

6.2.2 Discussion of Results. Table 1 provides quantitative results. As can be seen in Figure 7 as well, the *CA-Only* simulation variant clearly fails in this corridor because global planning is required. The *Standard* variant lets some agents pile up around an obstacle. This is reflected by high averages and standard deviations for *Time* and *SlowTime*. Also, parts of the corridor are under-utilized because

they are not chosen by global planning. When using the *Strategic* variant, agents make better use of alternative paths. Consequently, they spread out over the corridor, they walk faster on average, and they spend less time walking slowly. This shows that it can be useful to let agents take global detours based on their local velocities.

We have tested both a 'coarse' version of *Strategic* (with $T_R = 2$, $W = 1.2$, and *Strict* = `False`) and a 'fine' version (with $T_R = 0.5$, $W = 1.2$, and *Strict* = `True`). Both versions improve upon the *Standard* simulation, with only little differences between them. This suggests that very frequent re-planning is not strictly necessary, which is good news for the simulation's computation time.

Density-based global path planning [van Toll et al. 2012] has a similar purpose, but such a method does not recognize small congestions in large areas of the navigation mesh, and it does not consider the *directions* in which agents move. The method could be combined with ours to further improve the crowd's behavior.

6.3 Crowd Flows with Environmental Guidance

6.3.1 Experiment. In the same environment, we also simulate *bidirectional* flows by sending every other agent in the opposite direction. We observe deadlocks in all simulations, both for the *Standard* and *Strategic* variants (Figures 8(a) and 8(b)). Thus, the strategic level does not compensate for poor coordination between agents.

However, an interesting 'bonus' application of topological strategies is that we can now *guide* agents around specific obstacles in specific ways, by enforcing certain navigation decisions for these obstacles. Whenever an agent (re-)plans a global path, we include these obstacle decisions in the constraint strategy.

6.3.2 Discussion of Results. Figure 8(c) shows what happens in *UTurn* if we instruct agents to pass the blue-colored obstacles on the right during global path planning. Directing the crowd in this way prevents deadlocks in all simulation variants (except *CA-Only*).

As another example, Figure 9 shows a scenario in which 80 agents need to move to the opposite end of a circle with an inner diameter of 15m, while avoiding four 2×2m obstacles in the middle. In simulations without environmental guidance, some agents get stuck between the obstacles. However, when we instruct agents to pass these obstacles along the right, the problem is easily resolved.

Thus, navigation strategies provide an intuitive way to enforce decisions in the crowd. This type of environmental guidance is comparable to placing signage in a real-world environment to improve pedestrian flows. The decisions could also be made agent-dependent, e.g. to simulate how pedestrians respond to a sign when they see it. We plan to further investigate this in future work.

6.4 Performance Measurements

6.4.1 Experiment. We measure the performance of our system in the *UTurn* scenarios, for both the unidirectional flow and the bidirectional flow with guidance. Between $t = 50$s and $t = 300$s, we measure the average running time (over all frames) of each *simulation substep* (as described in Appendix A.1). We test both the 'coarse' and the 'fine' variant of *Strategic*, as defined earlier.

6.4.2 Discussion of Results. Figure 10 plots our results. In all simulations, computing visibility polygons (substep 2) is the most expensive substep, followed by collision avoidance (substep 5). Computing the local strategies S_C and S_F (substep 6) is about as costly as finding neighboring objects (substep 3). The strategic level (substep 7) is cheap in the coarse simulations. In the fine simulations, agents re-plan more often and they occasionally re-run collision avoidance with additional constraints. Generally, the performance of substep 7 depends on how often we allow agents to re-plan (which can be controlled by T_R), how often they actually re-plan in a certain scenario, and how long each re-planning query takes (which depends on the complexity of the environment).

Overall, the bidirectional fine simulation requires the most computation time, with an average of 4.8ms per frame. (Note: each frame simulates 100ms, so the simulation easily runs in real time.) This scenario contains 194 agents on average throughout our time window. These results suggest that our implementation can simulate over 4,000 agents in real time on 4 threads, although the exact limit depends on the scenario at hand. Most of the computation time is spent on visibility queries and collision avoidance.

For this paper, the most important observation is that the strategy-related substeps are not expensive, unless we let agents re-plan very often. Coincidentally, frequent re-planning does not necessarily improve the behavior of agents, as shown earlier.

7 DISCUSSION

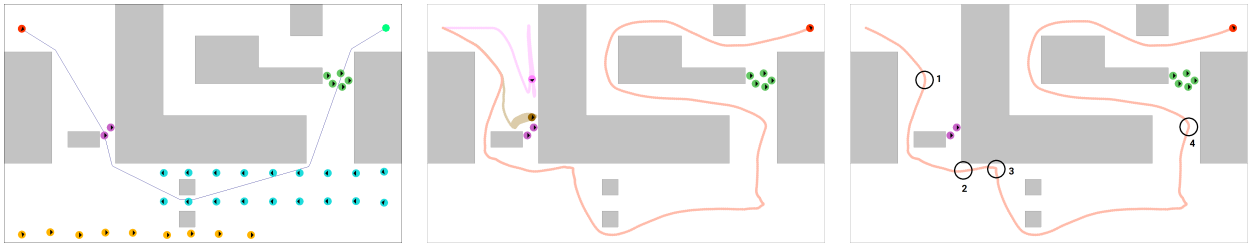
Our method offers a way to coordinate between the different levels of agent navigation. Still, the overall 'intelligence' of agents depends largely on the quality of the navigation algorithms themselves, and on the parameters within each algorithm. It can even be desirable to vary these settings per agent, to obtain a more varied crowd. Thus, it is difficult to draw general conclusions from any experiment. Our results at least indicate that topological strategies *can* improve a simulation. However, as is usual in this research area, each scenario may require careful visual inspection and parameter tuning.

On a related note, we can now detect strategic conflicts and find possible solutions, but choosing the right solution at the right time is difficult and highly scenario-dependent. Our suggested choice model is only one of many possibilities. Even within this model, each situation may impose its own optimal values of T_R and W.

Also, global and local planning are still two separate steps that use different information to determine a strategy. In theory, agents could end up switching back and forth between paths, because global planning 'forgets' any previous local constraints that have moved out of sight. Preventing such problems requires more advanced rules and memory models for the individual agents.

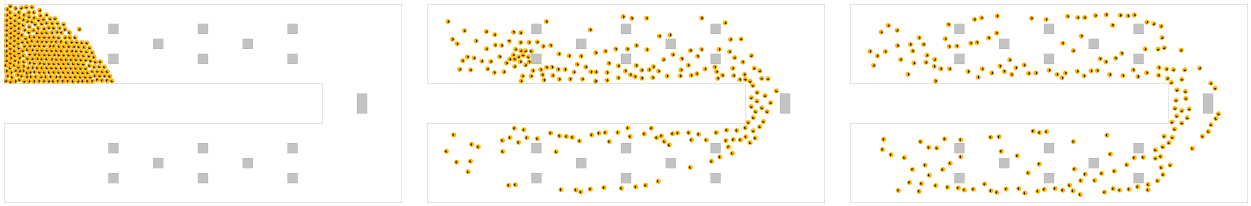
Agents can still get stuck in cases that are not detected by the strategic level. So far, we only handle obstacle-related conflicts between path following and collision avoidance. As an example of a problem that we cannot solve yet, imagine an agent that passes an obstacle and then notices a congestion *too late* (because it was out of sight for the collision-avoidance algorithm). Our strategic level does not yet recognize this, nor does it explicitly offer the agent the option to turn around. To solve problems such as this one, we need to add more navigation algorithms to the framework, and enrich these with the concept of topological decisions.

The framework from Section 5 can be augmented with many other navigation techniques, such as mid-term planning [Bruneau and Pettré 2017], detailed navigation in high-density crowds [Stüvel



(a) Overview (b) Trajectories (c) Re-planning actions for *Strategic*

Figure 6: Single-agent demonstration. (a) An agent is instructed to move from the top left to the top right. The curve is its initial global path. (b) The agent's trajectory using *CA-Only* (pink), *Standard* (brown), and *Strategic* (red, with $T_R = 2$, $W = \infty$, and *Strict* = `False`). (c) The *Strategic* agent changes its global path on several occasions, indicated by the black circles.

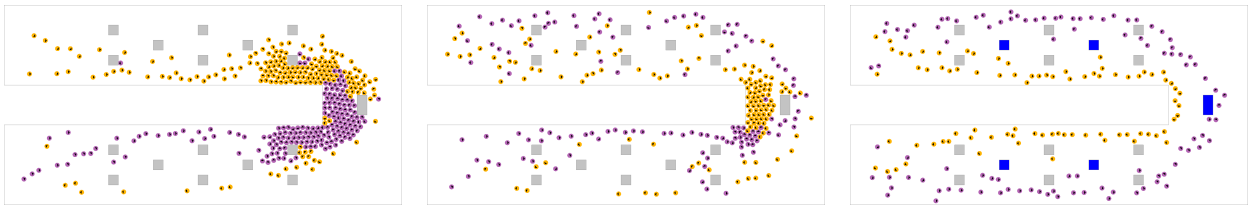


(a) *CA-Only* (b) *Standard* (c) *Strategic* ($T_R = 2$, $W = 1.2$, *Strict* = `False`)

Figure 7: Screenshots of unidirectional crowd flows in the *UTurn* environment, with different simulation variants. Agents are instructed to move from the top-left to the bottom-left.

Table 1: Quantitative results of the *UTurn* experiment from Section 6.2. Standard deviations are shown in square brackets.

Simulation variant	*NrAgents*	*Dist*	*Time*	*AvgSpeed*	*SlowTime*
Standard	513	72.31 [3.47]	76.71 [9.53]	0.96 [0.13]	5.30 [7.53]
Strategic ($T_R = 2$, $W = 1.2$, *Strict* = `False`)	555	74.11 [4.93]	63.83 [3.58]	1.16 [0.07]	0.12 [0.36]
Strategic ($T_R = 0.5$, $W = 1.2$, *Strict* = `True`)	556	73.95 [5.15]	64.17 [3.39]	1.15 [0.07]	0.10 [0.29]



(a) *Standard* (b) *Strategic* ($T_R = 2$, $W = 1.2$, *Strict* = `False`) (c) *Standard* + guidance

Figure 8: Screenshots of bidirectional crowd flows in the *UTurn* environment, with and without environmental guidance. The orange agents move from the top-left to the bottom-left; the purple agents move in the opposite direction.

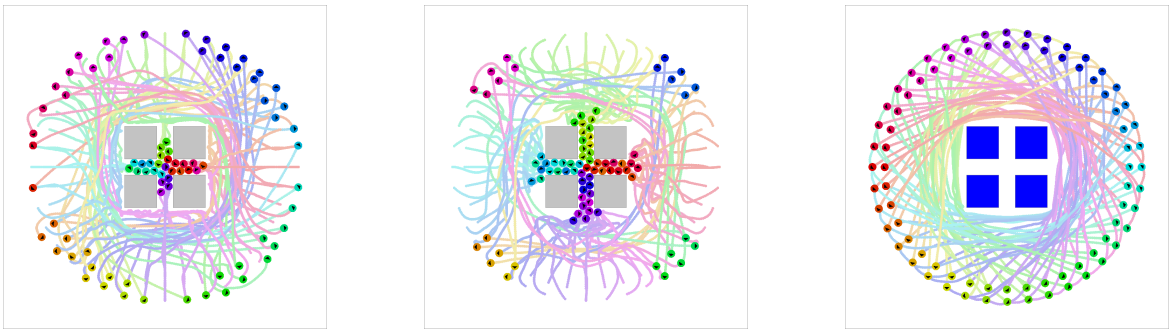


Figure 9: Results of a circle scenario using *CA-Only* (left), *Standard* (middle), and *Standard* with guidance (right).

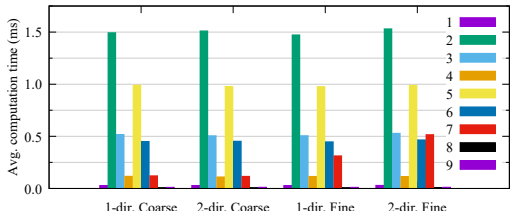


Figure 10: Performance of the *UTurn* simulations. Labels 1–9 refer to the simulation substeps described in Appendix A.

et al. 2016], and density-based adaptation of paths [van Toll et al. 2012] or velocities [Best et al. 2014]. It would not be meaningful to directly *compare* our work against such techniques because they all have different purposes. In fact, these algorithms can be *combined* with our work, so that each algorithm suggests its own navigation strategy for an agent. The strategic level can then be used to detect and resolve conflicts between these strategies.

We have not yet considered scenarios where agents can choose between *multiple goals*, such as an evacuation with multiple ways to leave a building. This can cause situations where an agent may not only update its path, but also change its goal. This is compatible with our method if we extend global path planning to use multiple possible goals. An agent will then automatically choose another goal when nearby obstacle constraints make it more attractive.

Finally, a largely open research question is how to measure the *realism* of a crowd simulation. Our method can solve navigation problems that are easy to see in a top-down view, and it results in trajectories that are objectively more efficient. However, in complex scenarios with obstacles, it is unclear how real humans behave, how to translate this to simulation settings, and how to properly compare a simulation to reality. Overall, the research area is not yet ready to make any general claims about how 'human-like' a simulation is.

8 CONCLUSIONS AND FUTURE WORK

We have presented a method for improving the navigation behavior of autonomous agents in virtual environments. We have defined a *navigation strategy* as a set of decisions to pass obstacles and agents on certain sides. Such a strategy can be obtained from the result of global navigation (path planning) *and* of local navigation (path following and collision avoidance). With this uniform definition, we can detect conflicts between the strategies produced by different algorithms. For example, when path following and collision avoidance disagree on how to pass an obstacle, an agent can attempt to re-plan its global path, using its local strategy as guidance.

This concept can be implemented as an additional 'strategic level' in existing crowd-simulation frameworks. Our experiments show that this can improve the behavior of agents in real time. The strategy concept is also suitable for explicitly sending agents in certain directions, e.g. to simulate the effect of directional signs.

As mentioned earlier, our simulation framework can be enhanced with many more navigation algorithms. In future work, we first intend to add mid-term planning and handle the agent-related conflicts between that and collision avoidance. Eventually, we envision a simulation in which agents can use many navigation algorithms for different occasions. The generic concept of a strategy makes it easier to detect when two algorithms behave inconsistently, or when certain algorithms are not sufficient for solving a problem.

To plan an alternative path, we currently run the A* search from scratch. We can improve efficiency by re-using parts of the old path, or perhaps by using a *hierarchical* approach in which the agent starts with a coarse path and refines it during the simulation.

Our experiment with environmental guidance (Section 6.3) suggests another promising direction for future work. We could extend the simulation so that agents update their global strategy over time, based on the directional hints that they observe. This could be used to study the effectiveness of signage in urban environments.

We consider this work to be a first step towards looking at agent-based crowd simulation in a fundamentally different way. By modelling agent navigation via topological decisions, we can bridge conceptual gaps between algorithms. On the long term, we hope that this insight will lead to a fully hybrid simulation technique, in which all aspects of agent navigation are merged into one process.

REFERENCES

Andrew Best, Sahil Narang, Sean Curtis, and Dinesh Manocha. 2014. DenseSense: Interactive crowd simulation using density-dependent filters. In *Proc. 13th ACM SIGGRAPH / Eurographics Symp. on Computer Animation*. 97–102.

Subhrajit Bhattacharya, Maxim Likhachev, and Vijay Kumar. 2012. Topological constraints in search-based robot path planning. *Autonomous Robots* 33, 3 (2012), 273–290.

Boost. 2019. The Boost C++ library. http://www.boost.org/.

Julien Bruneau and Julien Pettré. 2017. EACS: Effective Avoidance Combination Strategy. *Comput. Graph. Forum* 36, 8 (2017), 108–122.

Sean Curtis, Andrew Best, and Dinesh Manocha. 2016. Menge: A modular framework for simulating crowd movement. *Collective Dynamics* 1, A1 (2016), 1–40.

Teofilo B. Dutra, Ricardo Marques, Joaquim B. Cavalcante-Neto, Creto A. Vidal, and Julien Pettré. 2017. Gradient-based steering for vision-based crowd simulation algorithms. *Comput. Graph. Forum* 36, 2 (2017), 337–348.

Roland Geraerts. 2010. Planning short paths with clearance using Explicit Corridors. In *Proc. IEEE Int. Conf. Robotics and Automation*. 1997–2004.

Abhinav Golas, Rahul Narain, and Ming C. Lin. 2013. Hybrid long-range collision avoidance for crowd simulation. In *Proc. ACM SIGGRAPH Symp. Interactive 3D Graphics and Games (I3D '13)*. 29–36.

Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Systems Science and Cybernetics* 4, 2 (1968), 100–107.

Dirk Helbing and Péter Molnár. 1995. Social force model for pedestrian dynamics. *Physical Review E* 51, 5 (1995), 4282–4286.

John Hershberger and Jack Snoeyink. 1994. Computing minimum length paths of a given homotopy class. *Comput. Geom. Theory Appl.* 4, 2 (1994), 63–97.

Mario Höcker, Volker Berkhahn, Angelika Kneidl, André Borrmann, and Wolfram Klein. 2010. Graph-based approaches for simulating pedestrian dynamics in building models. In *eWork and eBusiness in Architecture, Engineering and Construction*. 389–394.

Norman S. Jaklin, Atlas F. Cook IV, and Roland Geraerts. 2013. Real-time path planning in heterogeneous environments. *Computer Animation and Virtual Worlds* 24, 3 (2013), 285–295.

Marcelo Kallmann. 2014. Dynamic and robust Local Clearance Triangulations. *ACM Trans. Graph.* 33, 5, Article 161 (2014), 17 pages.

Mubbasir Kapadia, Alejandro Beacco, Francisco Garcia, Vivek Reddy, Nuria Pelechano, and Norman I. Badler. 2013. Multi-domain real-time planning in dynamic environments. In *Proc. 12th ACM SIGGRAPH/Eurographics Symp. Computer Animation*. 115–124.

Ioannis Karamouzas, Roland Geraerts, and Mark H. Overmars. 2009. Indicative routes for path planning and crowd simulation. In *Proc. 4th Int. Conf. Foundations of Digital Games*. 113–120.

Peter M. Kielar, Daniel H. Biedermann, and André Borrmann. 2016. *MomenTUMv2: A modular, extensible, and generic agent-based pedestrian behavior simulation framework*. Technical Report TUM-I1643. Technische Universität München, Institut für Informatik.

Ross A. Knepper, Siddhartha S. Srinivasa, and Matthew T. Mason. 2011. Toward a deeper understanding of motion alternatives via an equivalence relation on local paths. *Int. Journal of Robotics Research* 31, 2 (2011), 167–186.

Angelos Kremyzas, Norman S. Jaklin, and Roland Geraerts. 2016. Towards social behavior in virtual-agent navigation. *Science China - Information Sciences* 59, 11 (2016), 112102.

Christoforos I. Mavrogiannis and Ross A. Knepper. 2019. Multi-agent path topology in support of socially competent navigation planning. *Int. Journal of Robotics Research* 38, 2–3 (2019), 338–356.
Ramon Oliva and Nuria Pelechano. 2013. NEOGEN: Near optimal generator of navigation meshes for 3D multi-layered environments. *Computers & Graphics* 37, 5 (2013), 403–412.
Sachin Patil, Jur P. van den Berg, Sean Curtis, Ming C. Lin, and Dinesh Manocha. 2010. Directing crowd simulations using navigation fields. *IEEE Trans. Vis. Comput. Graphics* 17 (2010), 244–254. Issue 2.
Nuria Pelechano, Jan M. Allbeck, and Norman I. Badler. 2007. Controlling individual agents in high-density crowd simulation. In *Proc. ACM SIGGRAPH/Eurographics Symp. Computer Animation*. 99–108.
Nuria Pelechano, Jan M. Allbeck, Mubbasir Kapadia, and Norman I. Badler. 2016. *Simulating Heterogeneous Crowds with Interactive Behaviors*. CRC Press.
Julien Pettré, Helena Grillon, and Daniel Thalmann. 2007. Crowds of moving objects: Navigation planning and simulation. In *Proc. IEEE Int. Conf. Robotics and Automation*. 3062–3067.
Sybren A. Stüvel, Nadia Magnenat-Thalmann, Daniel Thalmann, and A. Frank van der Stappen. 2016. Torso crowds. *IEEE Trans. Vis. Comput. Graphics* 23, 7 (2016), 1823–1837.
Avneesh Sud, Russell Gayle, Erik Andersen, Stephen Guy, Ming C. Lin, and Dinesh Manocha. 2007. Real-time navigation of independent agents using adaptive roadmaps. In *Proc. ACM Symp. Virtual Reality Software and Technology*. 99–106.
Daniel Thalmann and Soraia R. Musse. 2013. *Crowd Simulation* (2 ed.). Springer.
Adrien Treuille, Seth Cooper, and Zoran Popović. 2006. Continuum crowds. *ACM Trans. Graph.* 25 (2006), 1160–1168. Issue 3.
Jur P. van den Berg, Stephen J. Guy, Ming C. Lin, and Dinesh Manocha. 2011. Reciprocal n-body collision avoidance. In *Proc. 14th Int. Symp. Robotics Research*. 3–19.
Arthur van Goethem, Norman Jaklin, Atlas F. Cook IV, and Roland Geraerts. 2015. On streams and incentives: A synthesis of individual and collective crowd motion. In *Proc. 28th Conf. Computer Animation and Social Agents*.
Wouter van Toll, Norman Jaklin, and Roland Geraerts. 2015. Towards believable crowds: A generic multi-level framework for agent navigation. In *ASCI.OPEN*.
Wouter van Toll, Roy Triesscheijn, Roland Geraerts, Marcelo Kallmann, Ramon Oliva, Nuria Pelechano, and Julien Pettré. 2016. A comparative study of navigation meshes. In *Proc. 9th ACM SIGGRAPH Int. Conf. Motion in Games*. 91–100.
Wouter G. van Toll, Atlas F. Cook IV, and Roland Geraerts. 2012. Real-time density-based crowd simulation. *Computer Animation and Virtual Worlds* 23, 1 (2012), 59–69.
David Wolinski, Ming C. Lin, and Julien Pettré. 2016. WarpDriver: Context-aware probabilistic motion prediction for crowd simulation. *ACM Trans. Graph.* 35, 6, Article 164 (2016), 164:1–164:11 pages.

A IMPLEMENTATION DETAILS

In our implementation of the framework, we model agents as disks with a radius of 0.25m, a mass of 80kg, and a preferred speed s_{pref} of 1.2m/s, unless specified otherwise for a specific experiment.

A.1 Simulation Loop

Our simulation loop uses fixed timesteps (*frames*) of $\Delta t = 0.1$s. In every frame, the program performs the following *substeps*:

(1) Build a kd-tree of the agents' positions.
(2) Compute a visibility polygon for each agent's position, using the navigation mesh described in Section A.2.1.
(3) Compute the neighboring objects of all agents. For an agent A_j, the neighboring *agents* are those that currently collide with A_j, plus the 10 nearest agents that are in the visibility polygon of A_j *and* in a 180° cone centered at A_j's viewing direction $\mathbf{d}$. The neighboring *obstacles* are all (partial) obstacle segments within 5m of A_j that bound the visibility polygon.
(4) Perform path following for all agents (see Section A.2.2).
(5) Perform collision avoidance for all agents (see Section A.2.3).
(6) For each agent, convert the preferred velocity $\mathbf{v}_{\text{pref}}$ to a strategy S_F and the new velocity $\mathbf{v}_{\text{new}}$ to a strategy S_C.
(7) For each agent, re-plan if $\mathbf{v}_{\text{pref}}$ could not be computed. Otherwise, perform the strategic level described in Section 5.1 (i.e. check for strategic conflicts and respond to them).
(8) Compute contact forces [Helbing and Molnár 1995] for any collisions that are currently happening. This yields a force vector $\mathbf{F}$ for each agent.
(9) Update the positions of all agents as follows:

$$\mathbf{a} := \mathbf{F}/m + (\mathbf{v}_{\text{new}} - \mathbf{v})/t_r,\ \ \mathbf{v} := \mathbf{v} + \mathbf{a} \cdot \Delta t,\ \ \mathbf{p} := \mathbf{p} + \mathbf{v} \cdot \Delta t$$

where $t_r = 0.5$s is a *relaxation time*. The viewing direction $\mathbf{d}$ is updated similarly to $\mathbf{v}$, but without the factor $\mathbf{F}/m$, so that agents do not rotate when they are pushed aside.

All substeps except the first contain an independent process per agent, and their work can be distributed over parallel threads. We use 4 threads for our performance experiments in Section 6.4.

A.2 Navigation Algorithms

A.2.1 Path Planning. As our navigation mesh, we use the Explicit Corridor Map (ECM) [Geraerts 2010]: the medial axis (MA) of the environment annotated with nearest-obstacle information. We construct it using the Voronoi library of Boost [2019]. To obtain a set of passages from the ECM, we compute its *local minima*: the points on the MA where the distance to obstacles is locally smallest. For each local minimum p, we define a corresponding passage as the line segment between the two nearest obstacle points of p. There is at most one local minimum per MA edge, so there are $O(N)$ passages in total, and they subdivide the environment into $O(N)$ regions.

To plan a *path* using the ECM, we first use A* search [Hart et al. 1968] to find a shortest path on the medial axis. We then apply the funnel algorithm [Hershberger and Snoeyink 1994] to compute a shortest path (that keeps some distance to obstacles) in the same homotopy class. The sequence of *passages* traversed by this path can be obtained trivially during the A* search.

A.2.2 Path Following. We use the following path-following algorithm. Let $\mathbf{p}^-_{\text{ref}}$ and $\mathbf{p}^-_{\text{att}}$ be an agent's last used reference point and attraction point on its path π. First, we compute the new reference point $\mathbf{p}_{\text{ref}}$ as the point on π between $\mathbf{p}^-_{\text{ref}}$ and $\mathbf{p}^-_{\text{att}}$ that is closest to the agent [Jaklin et al. 2013]. Then, we compute the new attraction point $\mathbf{p}_{\text{att}}$ as the first point on π from the following options:

- the last point of π (i.e. the agent's goal);
- the point that lies 5m ahead of $\mathbf{p}_{\text{ref}}$ along π;
- the first point on π that the agent cannot see, starting at $\mathbf{p}_{\text{ref}}$.

If $\mathbf{p}_{\text{att}} = \mathbf{p}_{\text{ref}}$, then the agent does not know how to follow its path, and it will re-plan in substep 7. Otherwise, we compute the preferred velocity $\mathbf{v}_{\text{pref}}$ and the strategy S_F as described in Section 5.

A.2.3 Collision Avoidance. We use a custom collision-avoidance routine that yielded better behavior than existing methods. To compute a new velocity $\mathbf{v}_{\text{new}}$ for an agent A_j, we sample 15 candidate angles in a 180° range around $\mathbf{v}_{\text{pref}}$, and 2 candidate speeds (s_{pref} and $0.5 \cdot s_{\text{pref}}$). Each combination leads to a candidate velocity.

For a candidate velocity $\mathbf{v}''$, let $\delta(\mathbf{v}'')$ be the distance to the first collision with nearby objects if A_j would use $\mathbf{v}''$, clamped to a maximum distance $d_{\max} = 5$m. We choose the optimal velocity as:

$$\underset{\mathbf{v}''}{\operatorname{argmin}} \left(d_{\max} - \delta(\mathbf{v}'') + \angle(\mathbf{v}'', \mathbf{v}_{\text{pref}}) + \angle(\mathbf{v}'', \mathbf{v}) + \frac{||\mathbf{v}''|| - s_{\text{pref}}}{s_{\text{pref}}} \right)$$

where $\angle$ denotes the angle between two vectors (in radians), and $\mathbf{v}$ is the agent's last used velocity.